… Then We Can Build A Green Relational Database System.

As is stated in the bible for grad students: ***'Cracking the Coding Interview***,' Gayle Laakmann McDowell, 6th Edition., chapter 14 'Databases,' p. 172:

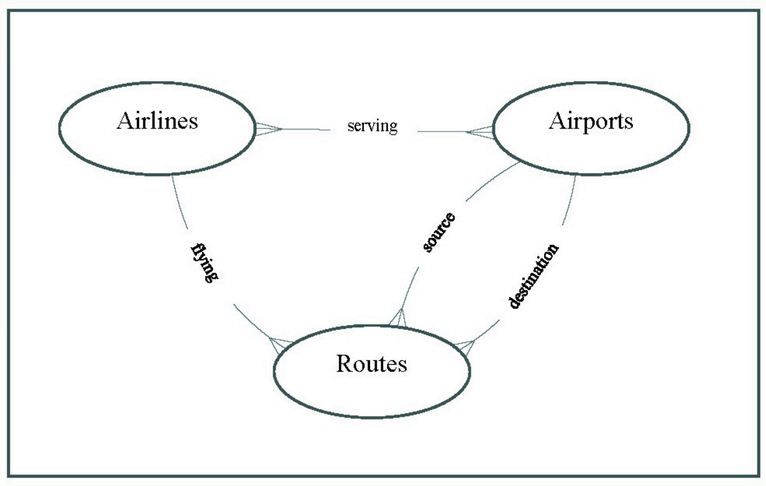
"Large Database Design: When designing a large scalable database, database joins (which are required in the above examples) are generally very slow, Thus, you must denormalize your data. Think carefully about how your data will be used - you'll probably need to duplicate data in multiple tables."

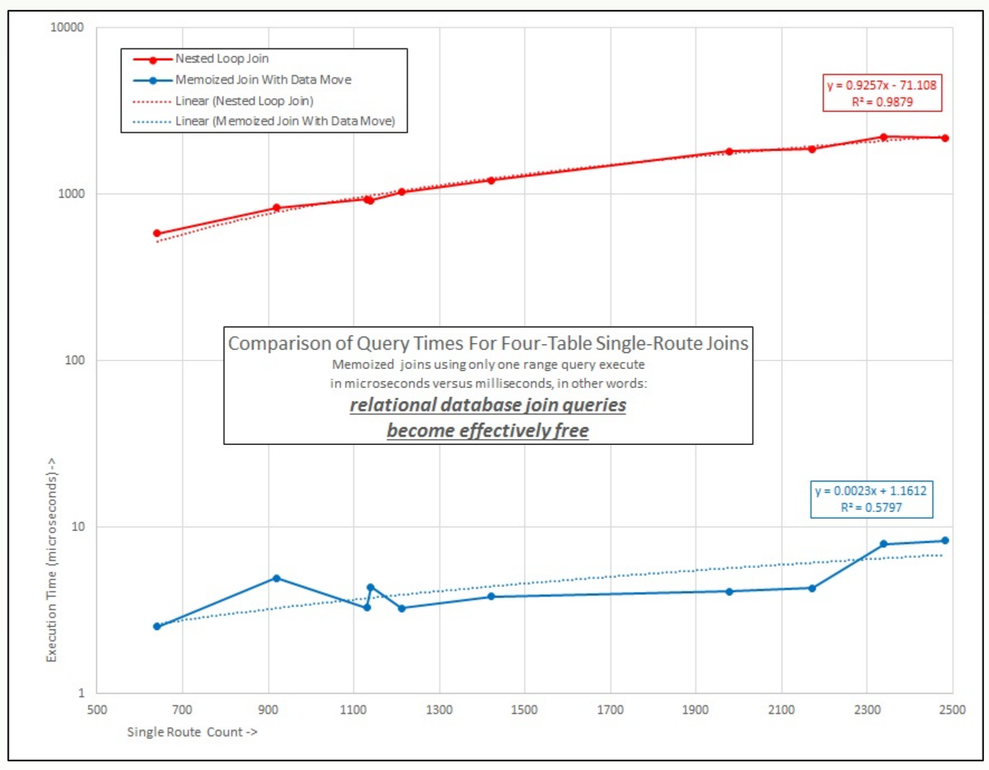
However, we can ignore that advice with enormous shared memory-centric systems like Gen-Z and Intel 3D XPoint and make relational database joins free, because any number of tables can be joined of any size accessing shrunken memoized joined row data to support queries executing in a microsecond or so in the data center, or remotely at the speed of the network:

1. **Using new faster string class objects that scale better:** Direct, Symbiont[+PMNK – stands for "Poor Man's Normalized Key" – Goetz Graefe https://doi.org/10.1145/1140402.1140409], and Head[+PMNK] instead of std::string.
2. **The new string classes have different functions:** for the relational database system, Direct becomes the basis for RowString (with columns) and Head becomes the basis for IndexString (friend class of RowString using PMNK).
3. **Using slab allocation:** instead of fine-grained allocators: using automatic variables on the stack, or mmap.
4. **Using Contiguous built-in arrays:** instead of standard library containers.
5. **Using Relation vectors:** describing from and to relations between columns of tables, in C++11 variadic template parameter packs as arguments to create joins.
6. **Using generic programming:** with recursion across those parameter packs at compile time, with constexpr initJoin and join routines leaving a runtime consisting of pointer fetches and some data movement.
7. **Using inference programming:** replacing the standard offline utilities and having SQL's DDL, DML and metadata optimized out of the C++11 runtime using const static globals and constexpr operations.
8. **Finally, using memoized joins:** replacing materialized views with 4% (for 100-byte rows) or less of the space complexity, but retaining full expressive access by inference using <tuple>.

The demo programs use the airport, airline and route data for the world's airlines:

1. **TableDemo** (walks the user through the evolution of the classes)
2. **JoinDemo** (showing nested loop 4-table join times that are less than 10 microseconds: where the program loads 3 tables from CSV files, builds ten indexes and performs 10 4-table joins in less than 1/8 of a second)
3. **BigJoinDemo** (showing 7-table memoized join times that are flat at 1.8 microseconds on a slow box) are there to show the function, and in debug build can show this performance is not sleight of hand.



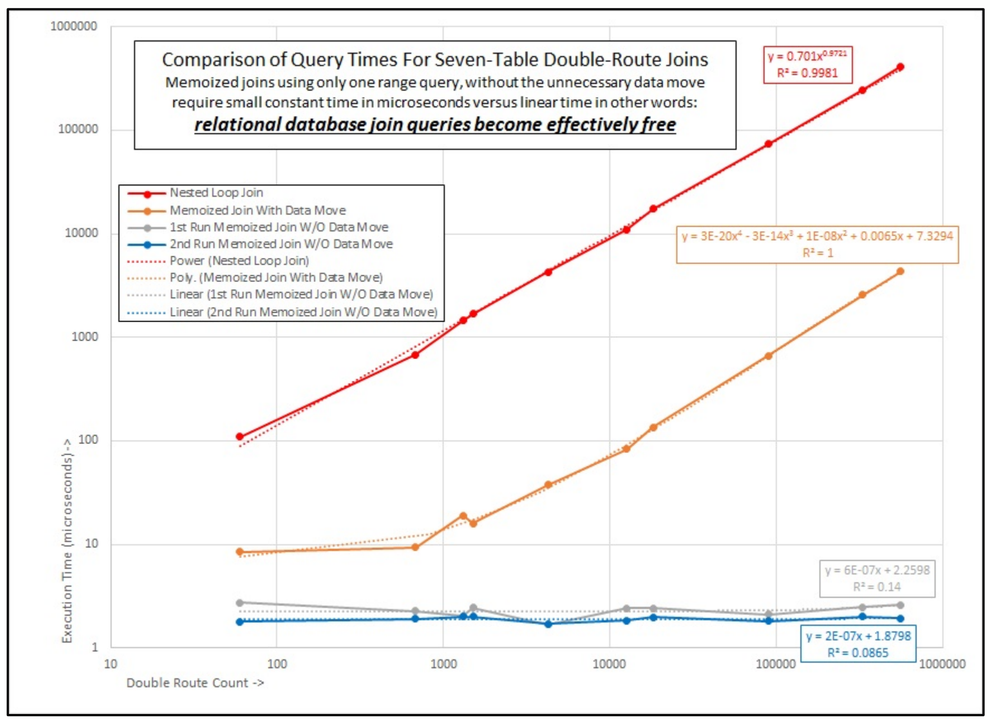


At a high level, there are a collection of variadic template classes:

1. The RowString class to hold table data
2. The RowString's friend class IndexString to index the columns in the rows
3. The RelationVector class structures the joins in ordered sets of directed column pairs called parameter packs in C++
4. The heavyweight QueryPlan class to hold the relational database metadata and optimize and construct the joins at compile time using the type system in a constexpr function called initJoin()
5. QueryPlan's extremely lightweight friend class JoinedRow which retrieves the query output from the QueryPlan join() constexpr function.

Future designs are discussed in the Technical White Paper on the GitHub:

1. BASE + ACID database systems where these fast BASE relational database joins can be built on-the-fly using snapshots as the ACID database is doing ingest
2. how to mutate the immutable relational database into a full ACID database support ultra-fast massive join queries without slowing ingest speeds, and
3. how to execute everything in parallel, scaling up AND out.



C. Johnson, email: cjohnson@member.fsf.org, GitHub: https://github.com/charlesone/Kaldanes